ABSTRACT
Component based software development (CBSD) becomes a popular paradigm for Internet based systems. Compared to other popular paradigms, CBSD supports the development from reusable components other than the development from the scratch. Consequently, modeling becomes more important than programming and the modeling techniques in traditional paradigms have to be changed more or less. Particularly, improper selection and misuse of modeling techniques would prevent the target system from benefiting from CBSD and even make the project fail. CBSD provides a bottom-up way by using existing middleware infrastructures, but this technology is not able to guide systematically the CBSD process, especially the component composition at higher abstract levels. On the other hand, Software Architecture (SA) provides a top-down approach to realizing component-based reuse, but mainly focusing on high-level design and analysis and thus not fully able to support the transformation or composition to form an executable application. Naturally, a promising solution is to combine the above two approaches to realize component-based systems in a systematic and automated manner. In that sense, SA should play a centric role in the whole software lifecycle, that is, SA description is used as the blueprint and middleware technology as the runtime scaffold for component composition, maintenance and evolution. To demonstrate the philosophy, we propose a software reuse methodology, which is called ABC (Architecture Based Component Composition).

Categories and Subject Descriptors
D.2.13 [Software Engineering]: Reusable Software – Reuse models, Domain engineering.

General Terms
Design, Documentation.

Keywords
Component, software architecture, feature model, UML

1. APPROACH OVERVIEW
Component based software development (CBSD) becomes a popular paradigm for Internet based systems. Compared to other popular paradigms, CBSD supports the development from reusable components other than the development from the scratch. Consequently, modeling becomes more important than programming and the modeling techniques in traditional paradigms have to be changed more or less. Particularly, improper selection and misuse of modeling techniques would prevent the target system from benefiting from CBSD and even make the project fail.

CBSD provides a bottom-up way by using existing middleware infrastructures, but this technology is not able to guide systematically the CBSD process, especially the component composition at higher abstract levels. On the other hand, Software Architecture (SA) provides a top-down approach to realizing component-based reuse, but mainly focusing on high-level design and analysis and thus not fully able to support the transformation or composition to form an executable application. Naturally, a promising solution is to combine the above two approaches to realize component-based systems in a systematic and automated manner. In that sense, SA should play a centric role in the whole software lifecycle, that is, SA description is used as the blueprint and middleware technology as the runtime scaffold for component composition, maintenance and evolution. To demonstrate the philosophy, we propose a software reuse methodology, which is called ABC (Architecture Based Component Composition).

Copyright is held by the author/owner(s).
ICSE ’06, May 20-28, 2006, Shanghai, China.
ACM 1-59593-085-X/06/0005.
applications, such as the information modeling of Olympic Games 2008 in Beijing, Credit Management System in China Min Sheng Bank, etc. In this tutorial, we will discuss how to model a component based system under the guide of ABC and with the help of ABC supporting tools.

2. FEATURE MODELING

It’s easy for CBSD to achieve all of its promises if it is applied to an application domain or a product line. Under the domain oriented context, the problem of requirements modeling has to face some new challenges. Domain requirements modeling involves exploring commonality and variation of domains, which makes it much complex since a domain often contains a set of applications, and thus requires mechanisms to capture commonality and variation. Domain requirements modeling aims at getting a set of reusable requirements, which requires a more powerful framework of requirements models to support the reuse of them. Feature-oriented approaches have been proposed to resolve these challenges. Most current domain requirements modeling methods use the feature model as the center model, together with other models, such as use case model and object model.

The feature-oriented approach to requirements modeling treat features as the basic entities in the problem space, and use features and relations between features (namely, feature model) to specify the problem space. A feature describes a software characteristic from user or customer views, which essentially consists of a cohesive set of individual requirements. Technically, to model requirements of component based systems in feature models supports software architecting at the requirements level as well as facilitates the reuse of SA.

DETool, one of the ABC toolset, provides feature models for specifying requirements in a visual way. It builds a framework to model the domain requirements from five aspects, which also provides some basic constructs, including whole-part associations for organizing features, optional features and dimensional values for representing variations, propositional logics for constraining and validating variations.

3. ARCHITECTURE MODELING

In the phase of software architecting, the requirements specifications are refined, and some overall design decisions are made. To produce SA meeting functional and non-functional requirements of the target system, the architects may study the requirement specifications, refine the features and their relations in the problem space into conceptual components and connectors, and create necessary artificial components and connectors, produce dynamic and static SA models, build mapping relationships between requirement specifications and SA, check SA and so on. Particularly, since some pre-fabricated components may be reused, the architects will take the reusable components into consideration.

One of the most challenging issues of component based systems is how to survive in the rapid and continuous changes in the era of Internet. Middleware, the most popular runtime infrastructure for component based systems, provides powerful and practical mechanisms for monitoring and changing runtime systems. However, these mechanisms only handle “how to do” other than “why, when and what to do” the adaptation. Then the adaptation for changes is still difficult, error-prone and time-consuming. On the other hand, designers can predict some changes and plan corresponding adaptations but they do not take the runtime adaptable mechanisms into account. In our approach, self-adaptive software architecture (SASA) is proposed. In details, we synthesize some sophisticated methods in architecture based software engineering, that is, the quality analysis in software architecture for WHEN to change, the design and description of dynamisms in software architecture for WHAT to change, and the runtime software architecture for HOW to change.

ABC/ADL is an XML-based architecture description language and provides plentiful modeling constructs including styles, components, connectors, configurations, composite components, complex connectors, aspects and semantic descriptions. All of these elements can be modeled in a visual way with ABCTool, which also supports multiple views of architectural models, such as type views, process views, configuration views and adaptation views. Finally, ABCTool can transform SA elements to UML elements so that the components unavailable in the reusable assets repositories can be implemented in the object oriented paradigm.

4. OBJECT ORIENTED MODELING

Ideally, a component based system can be constructed by plentiful reusable components. However, it’s still far away from practice mainly because of the immature component market. So lots of components have to be implemented after architecture modeling. Considering the popularity of object oriented paradigm and the closed relation between objects and components, a component is usually implemented as a set of objects. It should be noted that it’s not sufficient for the existing object oriented modeling techniques to deal with the commonality/variability identification and adaptability as mentioned above.

JBOO is a visual modeling tool and compliant with UML 2.0, which is the latest version of the dominant object oriented modeling language. JBOO provides many mechanisms defined in Model Driven Architecture (MDA), which supports automatic code generation and reuse of design models. Particularly, JBOO can be seamlessly integrated with JBCL, a component library, for supporting the systematic component based development.

5. ACKNOWLEDGMENTS

We would like to thank our research groups who contribute so much to the design of the methods, the implementation of the tools, and plentiful case studies. This work was supported by the National Basic Research Program (973), the National Natural Science Foundation of China and the State 863 High-Tech Program. Many thanks to the authors of all references that cannot be listed here due to the space limit. Finally, useful materials can be found at http://www.sei.pku.edu.cn/en.